Computing and Informatics, Vol. 34, 2015, 254–274

IMPROVING THE PERFORMANCES OF ASYNCHRONOUS SEARCH ALGORITHMS IN SCALE-FREE NETWORKS USING THE NOGOOD PROCESSOR TECHNIQUE

Ionel Muscalagiu

The "Politehnica" University of Timisoara
The Faculty of Engineering of Hunedoara, Revolutiei, 5, Romania
e-mail: ionel.muscalagiu@fih.upt.ro

Horia Emil Popa, Viorel Negru

The University of the West
The Faculty of Mathematics and Informatics
Timisoara, V. Parvan 4, Romania
e-mail: {hpopa, vnegru}@info.uvt.ro

Abstract. The scale-free graphs were proposed as a generic and universal model of network topologies that exhibit power-law distributions in the connectivity of network nodes. In recent years various complex networks were identified as having a scale-free structure. Little research was done concerning the network structure for DisCSP, and in particular, for scale-free networks. The asynchronous searching techniques are characterized by the occurrence of nogood values during the search for a solution. In this article we analyze the distribution of nogood values to agents and the way how to use the information from the nogood; that is called the nogood processor technique. We examine the effect of nogood processor for networks that have a scale-free structure aiming to develop search algorithms specialized for scale-free networks of constraints, algorithms that require minimum costs for obtaining the solution. We develop a novel way for distributing nogood values to agents, thus obtaining a new hybrid search technique that uses the information from the stored nogoods. The experiments show that it is more effective for several families of asynchronous techniques; we perform tests with the model running on a cluster of computers. Also, we examine the effect of synchronization of agents' execution and of processing messages by packets in scale-free networks.

Keywords: Agents, distributed constraint programming, asynchronous search techniques, scale-free networks, nogood messages

Mathematics Subject Classification 2010: 68T42

1 INTRODUCTION

Constraint programming is a programming approach used to describe and solve large classes of problems such as searching, combinatorial and planning problems. Distributed constraint satisfaction problem (DisCSP) is a constraint satisfaction problem in which variables and constraints are distributed among multiple agents. This type of distributed modeling appeared naturally for many problems for which the information was distributed to many agents (distributed resource allocation problems, distributed scheduling problems, multi-agent truth maintenance tasks, etc.) [17, 10]. The idea of sharing various parts of the problem among agents that act independently and collaborate in order to find a solution by using messages has led to a formal problem known as the distributed constraint satisfaction problem (DisCSP) [17, 10]. DisCSPs are composed of agents, each has its local constraint network. Variables in different agents are connected by constraints forming a network of constraints. Agents must assign values to their variables so that all constraints between agents are satisfied.

Distributed networks of constraints have proven their success when modeling real problems. Many problems in the computer science area, engineering, biology can be modeled efficiently as constraint satisfaction problems (or distributed CSP). Some examples include: spatial and temporal planning, qualitative and symbolic reasoning, diagnosis, decision support, hardware design and verification, real-time systems and robot planning, protein structure prediction problem, DNA structure analysis, timetabling for hospitals, industry scheduling, transport problems [5], etc.

There are complete asynchronous searching techniques for solving the DisCSP in this constraints network, such as the ABT (asynchronous backtracking), AWCS (asynchronous weak commitment) [17, 18], DisDB (distributed dynamic backtracking) [6], ABTDO (dynamic ordering for asynchronous backtracking) [10].

In recent years various complex networks have been identified as having a scale-free structure [2, 3, 4]. Scale-free networks are abundant in nature and society, describing such diverse systems as the Internet, a network of routers connected by various physical connections, the chemical network of a cell, etc. Little research was done concerning the behavior of the search techniques in networks of constraints that have a structure of the scale-free networks type [16, 15]. Thus, few things are known about choosing the optimal search technique for topological structures of the scale-free network type. The purpose of the article is to develop search algorithms specialized for scale-free networks of constraints, algorithms that require minimum costs for obtaining a solution.

In the distributed constraint satisfaction area, the asynchronous weak commitment search algorithm (AWCS) [17, 18] plays a fundamental and pioneer role among algorithms for solving the distributed CSPs. The algorithm is characterized by an explosion of the nogood values, but by dynamically changing the agents' order an efficient algorithm is obtained.

The occurrence of nogood values has the effect of inducing new constraints. Nogood values show the cause of failure and their incorporation as a new constraint will teach the agents not to repeat the same mistake. The non-restriction for recording the nogood values could become, in certain cases, impracticable. The main reason is that the storing of nogood values excessively consumes memory and could lead to lowering the memory that has been left. Another unfortunate effect of storing a large number of nogood values is related to the fact that the verification of the current associations in the list of nogood values that are stored becomes very expensive; the searching effort removes the benefits brought by the nogood values storing. These elements are analysed, aiming to see if this nogood processor technique brings benefits in terms of efficiency.

In [1] the notion of a nogood processor is introduced for the first time. In [12] we try to adapt the nogood processor technique for the AWCS technique. This technique consists of storing the nogood values and further uses the information given by nogoods in the process of selecting a new value for the variables associated to agents.

In this paper we examine the effect of nogood processor for constraints networks of the scale-free type. We develop a novel way for the distribution of nogood values to agents, the experiments show that it is more effective for several families of asynchronous techniques. Starting with the results from [12], this study tries to adapt the version of nogood processor with the learning techniques in the purpose of finding a solution that increases the performance of the AWCS technique. The aim of these studies is to develop search algorithms specialized for scale-free networks.

The asynchronous search techniques can be characterized by the existence of a very large number of elements that can be introduced, without affecting the completeness of the algorithm. For example, processing the messages in packets or individual, storage or not of the nogood messages, message filtering, the priority order of the agents or synchronization of the agents' execution. All these elements influence the behavior of the search techniques, for example, the increase or decrease of the message flow, the quantity of constraints checked, etc. Thus, a correct evaluation of the performances presumes also the analysis of these elements. In this article we focus on two such elements in the case of the problems with a scale free network structure: processing the messages by packets and the opportunity of synchronizing the process of execution of the agents.

Specifically, it is interesting to investigate the opportunity of synchronizing the agents in case of asynchronous techniques for topological structures of the scale-free network type. The behaviors of several asynchronous techniques are investigated in two cases: the agents execute asynchronously the processing of received messages (the real-life situation) and the synchronous case where the agents' execution is

synchronized. In other words, the agents perform a computing cycle in which they process messages (in packets) from a message queue in the synchronous case. After that, a synchronization is done waiting for the other agents to finalize the processing of their messages. Combining all these elements allowed the identification of a hybrid search technique that has better performances for the problems with a scale-free network structure.

Previous research [8, 13] shows that synchronization is beneficial for some techniques, but increases the costs for others.

The evaluation of the performances of the AWCS technique is done using NetLogo. NetLogo is a programming environment with agents that allows the implementation of the asynchronous search techniques ([19], [20]). In order to make such estimation, the AWCS technique with nogood processor is implemented in NetLogo, using the models proposed in [11, 14], model named DisCSP-NetLogo. Implementation examples for the AWCS family can be found on the website [20]. For the first time, the experiments are done with a large number of agents (500 and 1 000), using a NetLogo model that runs on a cluster [14, 21].

2 THE FRAMEWORK

This section presents some notions related to the DisCSP modeling and AWCS algorithm [17, 18, 10].

Definition 1. The model based on constraints – constraint satisfaction problem (CSP), existing for centralized architectures, is defined by a triple (X, D, C), where: $X = \{x_1, \ldots, x_n\}$ is a set of n variables; whose values are taken from finite domains $D = \{D_1, D_2, \ldots, D_n\}$; C is a set of constraints declaring those combinations of values which are acceptable for variables.

The solution of a CSP implies to find an association of values for all the variables that satisfies all the constraints.

Definition 2. A problem of satisfying the distributed constraints (DisCSP) is a CSP, in which the variables and constraints are distributed among autonomous agents that communicate by exchanging messages. Formally, DisCSP is defined by a 5-tuple (X, D, C, A, ϕ), where X, D and C are as before, $A = \{A_1, \ldots, A_p\}$ is a set of p agents, and $\phi : X \longrightarrow A$ is a function that maps each variable to its agent.

In this article we will consider that each agent A_i has allocated a single variable x_i, thus $p = n$. Also, we assume the following communication model [17, 18]:

- Agents communicate by sending messages. An agent can send messages to other agents iff the agent knows the addresses of the agents.
- Delay in delivering a message is finite, although random. For transmission between any pair of agents messages are received in the order in which they were sent.

Asynchronous search algorithms are characterized by the agents using the messages during the process of searching for the solution. Typically, it uses two types of messages:

- *Ok* message, which contains an assignment variable-value and is sent by an agent to the constraint-evaluating-agent in order to see if the value is right.
- *Nogood* message, which contains a list (called nogood) with the assignments wherefore a looseness was found, is sent in case the constraint-evaluating-agent finds an unfulfilled constraint.

The *ordering* of agents is based on their priority, so that agents that are later in the ordering are termed "lower priority agents" [6, 17, 10]. Orderings of agents can be either static or dynamic. If agents are ordered before the start of the run of the search algorithm and the order is not changed during the run of the algorithm, it is called static. Some asynchronous search techniques use a static order among agents (e.g. the lexicographic order), order that is fixed before the beginning of the algorithm. Other asynchronous search techniques use a priority based order, that can change during the algorithm runtime (AWCS, ABT-DO). For example, the asynchronous weak commitment search (AWCS) algorithm implements a specific dynamic ordering heuristic.

Definition 3. Two agents are connected if there is a constraint among the variables associated to them. Agent A_i has a higher priority than agent A_j if A_i appears before A_j in the total ordering. Agent A_i is the value-sending agent and agent A_j the constraint-evaluating agent.

Definition 4. The *agent-view* list belonging to an agent A_i is the set of the newest associations received by the agent for the variables of the agents to which it is connected.

Definition 5. The *nogood* list is a set of associations for distinct variables for which an inconsistency was found (an unsatisfied constraint). The *agent-view* list together with the stored *nogood* values constitutes the working context of each agent, depending on them the agent makes decisions.

Definition 6. The *nogood* list received by agent A_i is consistent for that agent, if it contains the same associations as *agent-view* for all the variables of the parent agents A_k connected with A_i.

The AWCS algorithm [17, 18] is a hybrid algorithm obtained by the combination of ABT algorithm with WCS algorithm, which exists for CSP. It can be considered to be an improved ABT variant, but not necessarily by reducing the nogood values, but by changing the priority order. It deliberately follows to record all the nogood values to ensure the completeness of the algorithm, but also to avoid some unstable situations.

The authors show in [17] that this new algorithm can be built by a dynamical change of the priority order. The AWCS algorithm uses, like ABT, two types of ok and nogood messages, with the same significance. There is a major difference in the way it treats the ok message. In case of receiving the ok message, if the agent cannot find a value to its variable that should be consistent with the values of variables that have a greater priority, the agent not only creates and sends the nogood message, but also increases the priority in order to be maximum among the neighbors.

3 SCALE-FREE NETWORK

The study of complex network topologies across many fields of science and technology has become a rapidly advancing area of research in the last few years. One of the key areas of research is understanding the network properties that are optimized by specific network architectures. The last few years have led to a series of discoveries that uncovered statistical properties that are common to a variety of diverse real-world social, information, biological, and technological networks.

In recent years various complex networks have been identified as having a scale-free structure [2, 3, 4]. Scale-free networks are abundant in nature and society, describing such diverse systems as the Internet, a network of routers connected by various physical connections, SNS, the chemical network of a cell.

Not all nodes in a network have the same number of edges. The spread in the node degrees is characterized by a distribution function P(k), which gives the probability that a randomly selected node has exactly k edges. Since in a random graph the edges are placed randomly, the majority of nodes have approximately the same degree. One of the most interesting developments in understanding complex networks was the discovery that for most of large networks the degree distribution significantly deviates from a Poisson distribution. In particular, for a large number of networks, including the World Wide Web, the Internet or metabolic networks, the degree distribution follows a power-law for a large number of nodes [2, 3, 4]. Such networks are called scale free [3, 4].

A scale-free network is characterized by a power-law degree distribution as follows:

$$p(k) \propto k^{-\gamma} \tag{1}$$

where k is the degree and γ is the exponent that depends on each network structure. Scale-free networks have no scale because there is no typical number of links [3, 4]. The random network models assume that the probability that two nodes are connected is random and uniform. In contrast, most real networks exhibit some preferential connectivity.

4 IMPROVING THE PERFORMANCES OF ASYNCHRONOUS SEARCH ALGORITHMS USING NOGOOD PROCESSOR

4.1 Nogood Processor in Scale-Free network

A DisCSP can be represented by a constraint graph $G = (X, E)$, whose nodes represent the variables and edges represent the constraints:

1. $X = \{x_1, \ldots, x_n\}$ is a set of n nodes/variables
2. $E = \{(x_1, x_j)\}$ is a set of edges for which we have a constraint between variables x_i and x_j.

In the scale-free networks a few nodes concentrate many more connections than the rest which have just a few connections each. Starting from this observation we define the notion of a hub:

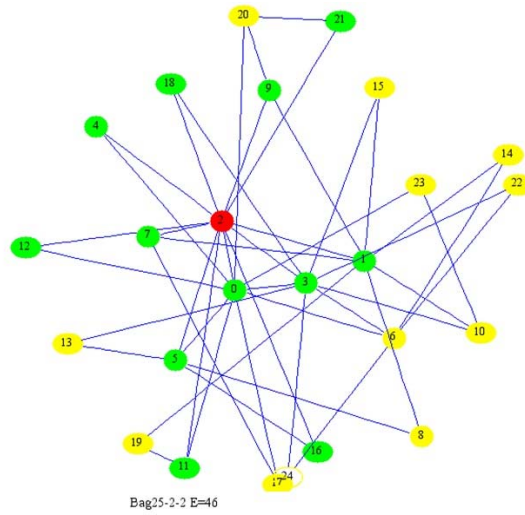
Definition 7. A node is called *hub* if it has a larger number of connections than a constant $c \in N$. Let H be set of hubs $H = \{x_i \mid x_i \in N, \deg(x_i) \geq c\}$ where $\deg(x_i)$ is the degree of node x_i.

Let us show a simple example. A constraint network of a DisCSP having a scale-free network structure, with 25 nodes and the minimal degree 2, is represented in Figure 1 a). If we consider the constant $c = 8$, we can remark the existence of a number of 4 hubs: $H_1 = 2$ (degree 12), $H_2 = 0$ (degree 10), $H_3 = 1$ (degree 9) and $H_4 = 3$ (degree 8). For nodes H_1, H_2, H_3, H_4 we can remark that they have many more connections compared to the rest, that have very few connections (see also Figure 1 b)).

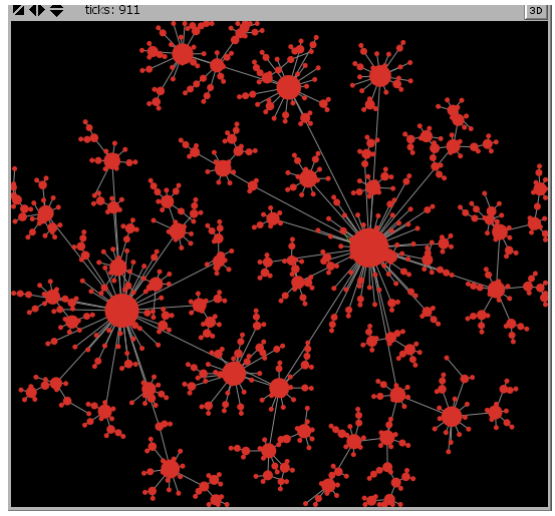
Definition 8. The *nogood processor* is a special agent associated to each agent A_i, that stores the received nogood values. Such an agent is called each time when the assignment of a new value is tried. Each nogood processor builds a database with the received nogood values (nogood store).

Those values may or may not be shared with the other nogood processors. There are two ways of sharing the values from the nogood store: through messages sent by agents of the nogood processor type, or through accessing the shared memory. Later, when an agent has found a candidate assignment for its variables, it consults the nogood processor to make sure that its assignment along with any known assignments of higher priority agents do not constitute a nogood.

In the AWCS algorithm when a nogood message is received by the agent A_i, the agent adds the nogood value to its nogood store and executes a verification of the inconsistencies for nogood. If the new nogood has also an unknown variable, the agent needs to receive the value from the corresponding agent. Unfortunately, the information from nogood is not completely used in the case of assigning a new value for the variable associated to the agent. It is possible that the nogood values contain a reference to this value, implying that the assignment has appeared before



a)



b)

Figure 1. Constraint network representing a DisCSP which has a scale-free network structure: a) A scale-free network with 25 nodes, b) A scale-free network with 900 nodes

as inconsistent. The use of this information will be the basis of the nogood processor technique construction.

In this paper it is considered that some agents have access to the results of their own nogood processor. More, each agent sends (stores) nogood values that it has received to the associated nogood processor. The information stored by each nogood processor will be used in searching a new value for each variable cared for by the agent. For this, each nogood processor will verify (asked by an agent) through its subroutine check-inconsistent-value-nogood-processor, if the value selected by the agent had no previous existence associated with the higher priority agents values. In Figure 2 we present, in the NetLogo language, the function of type *to-report* that returns certain values, for checking the inconsistency of a new value [12].

```
to-report check-inconsistent-value-nogood-processor [A_i]
    foreach Nogood ∈ nogoods-store [
      foreach x ∈ Nogood with the current priority from current-view
          * bigger than the agent's A_i [
        pos ⟵ position x in Nogood
        if x != item pos current-value [
          return consistent
        ]
      ]
      if curent-value! = item A_i in Nogood [
        return consistent
      ]
    ]
    return inconsistent
end
```

Figure 2. The Function check-inconsistent-value-nogood-processor

Another question needing an answer was identifying how the nogood processors are distributed. Practically, each agent, when receiving a nogood, sends this to an associated nogood processor that stores it into a nogood-store list (nogood processor only saves those new nogood values, eliminating the copies). The information will be used later when searching for a new value. Therefore the check-agent-view procedure will select a new value consistent with the agent view list and with the nogood list stored by the nogood processor (the value will be selected if, complementarily, the check-inconsistent-value-nogood-processor subroutine will return the consistent value).

The check-inconsistent-value-nogood-processor routine is used just for the higher priority agents, priority considered at the moment of nogood value storing. To put it differently, the identification of the agents with higher priority towards agent A_i, is not made by using their actual priority (in current-view), but the priority stored by the nogood processor.

4.2 Messages Management – Processing Messages by Packets

The asynchronous techniques are characterized by the existence of some message processing routines. We have treatment procedures for ok (info) messages, nogood (back), add or remove. Those routines treat sequentially the existing messages from the message queues. Usually, each agent extracts a message from its message queue, identifies the message type and calls the appropriate processing routine.

In [17, 18] the AWCS technique is presented, without detailing the manner in which the messages are processed, sequential or by packets, which is a protocol defining the order in which the messages should be treated.

In this paper we analyze implementations of the AWCS family with complete processing of messages: each agent treats entirely the existing messages in its message queue. Two solutions for complete processing of the queues of messages are evaluated. Figure 3 shows the new handle-message procedure, presented in the NetLogo language.

```
to handle-message [msize]
  set nrm 0
1 while [not empty? message-queue and nrm <= msize] or
1' while [not empty? message-queue] ***
 [
    set msg retrieve-message
    if (type msg = "ok?")
    [ Update agent-view with msg ]
    if (type msg = "nogood")
    [ handle-nogood-message msg ]
     ...
    set nrm nrm + 1
 ]
 if (nrm! = 0 )
   [Check-agent-view]
end
```

Figure 3. The handle-message for the AWCS

The new handle-message procedure in Figure 3 is applied by each agent to its message queue. The messages will be extracted in the order of arrival in the structures of queue type associated to the agents and treated as follows:

1. In the case that the message is of the ok? type, the working environment is updated with the received value, without checking the consistency of the existing values or trying a new value.
2. If the message is of the nogood type, the nogood value is saved and the working environment is updated with the values of the new agents unconnected initially with the current agent.

After scanning and extracting the messages completely or partially, the check-agent-view routine is called only once, for checking the consistencies from the agent-view list and the selection (eventually) of a new consistent value.

Concerning the complete or partial processing of the messages, it can be done by means of the msize variable or renouncing of the second condition of package limitation (the line labeled with ***). That has a role to decide the number of extracted and processed messages from the message queue. If msize is equal to the number of elements from the message queue (msize = length(message-queue)) or if that condition is missing, the procedure in Figure 3 allows the processing of all the messages. But, if msize is 1, the sequential processing of the messages is obtained.

The first variant supposes the insertion of line 1 instead of line 1'. In that case, each agent stops at the moment when either it has no more messages or msize messages were processed. The second variant supposes the insertion of line 1'. Message processing supposes an effort and thus a delay occurrence. It is possible that other messages arrive beside those from the initial moment. In the case of the second variant if later new messages appear, those are still treated thus surpassing the number msize of messages allowed. In this paper we implement both variants.

4.3 The Synchronization of the Agents' Execution

The asynchronous search techniques are search algorithms that run asynchronously in distributed systems. In reality, the agents run concurrently and asynchronously, each agent treating its messages from the messages queue in the arrival order, without waiting for the finalization of computations from the other agents. The analysis of experimental results shows that the AWCS techniques behave better in case of synchronizing the agents' execution [13].

In the majority of cases in which performances of search techniques were analyzed it was done on synchronous distributed systems. Because of simplicity reasons these distributed systems are usually simulators [17, 18, 10]. A synchronous distributed system is one of possible distributed systems, where all processes (agents) run their cycles synchronously. One cycle consists of activities so that all agents read incoming messages, do their local computation, and send messages to relevant agents. This article presents an opportunity for synchronizing the agents' execution in case of the AWCS family, but identifies problems with a scale-free network structure.

We investigate two variants of synchronization of the process of the agents' execution. The first method of synchronization is based on NetLogo elements, using the *ask-concurrent* command for executing the procedures for treating the agents messages. This command performs a synchronization of the commands attached to the agents so that the synchronization of the execution of agents is made automatically. Of course, each agent works asynchronously with the messages, but at the end of a command execution there is a synchronization of agents' execution. The simulated solution is based on the existence of a central agent (called in Netlogo Observer) which performs the synchronization of the agents' execution process.

In reality, the agents run concurrently and asynchronously, each agent treating its messages from the messages queue in the arrival order, without waiting for finalization of computations from the other agents. There is no central agent that can deal with the agents' synchronization. Therefore a second method is investigated in the context of using the nogood processor technique. The second solution from [13] consists of synchronizing only the neighboring agents. Each agent will wait for its connected neighbors to finish their computations which are placed before him in a lexicographical order. That solution allows a partial synchronization of the execution of agents. That second solution of partial synchronization is based on the use of a synchronization message. This message is similar to a token that each agent needs to receive in order to carry on with the execution of its computing cycle. For that, each agent uses a second communication channel for receiving synchronization messages (the first channel is used for receiving the ok or nogood messages).

The working protocol supposes for each agent the completion of two stages:

- each agent processes all the messages from its main communication channel performing a computing cycle. The moment when the main message channel is empty, it sends a message of the "synchronous" message type to the neighbouring agents, that are before him in a lexicographical order.
- after each cycle, the agent checks if it has received the synchronization messages from all of its neighbours, placed after him in a lexicographical order, and if not, it waits until it receives all the messages.

5 EXPERIMENTAL RESULTS

In this section we will present our experimental results obtained by implementing and evaluating the family of asynchronous techniques AWCS in NetLogo [19, 20]. The implementation and evaluation is done using the two models proposed in [11, 14]. The Netlogo implementations were run on a cluster of computers using RedHat Linux. The cluster allowed running instances of 500 and 1 000 agents, with various difficulties, without GUI, but with the synchronization of the agents' execution.

In order to investigate the effect of the synchronization (compared with the real situation of asynchronous running), several implementations were run, with GUI, on a Red Hat Linux system, but with a much lower number of agents (nodes = 100).

In this paper, the Java program developed by Sun Microsystems Laboratories is used as a scale-free network formation tool [7]. This program can generate scale-free networks given the number of nodes and the minimal degree of each agent (md). Scale-free networks are generated by the tool with the following parameters:

- nodes = 100, md = 16 and $\gamma = 1.8$
- nodes = 500, md = 4 and $\gamma = 1.8$
- nodes = 500, md = 16 and $\gamma = 1.8$
- nodes = 1 000, md = 2 and $\gamma = 2.1$.

We examine the performance of AWCS in scale-free networks. Specifically, we implemented and generated in NetLogo both solvable and unsolvable problems that have a structure of scale-free networks. Implementation examples for the scale-free network instance generator (using scale-free networks from [7]) can be found on the website [20]. We set the domain size of each variable to ten, i.e., domain = 10 which means $|D_i| = 10$. For the evaluations, we generate five scale-free networks. For each network, the constraint tightness varies from 0.1 to 0.9 by 0.1. For each constraint tightness, 200 random problem instances are generated. Thus, the results represent averages of these 1 000 instances for each of the five networks. These problems have a number of variables with a fixed domain. This creates problems that cover a wide range of difficulty, from easy problem instances to hard instances (for each version we are retaining the average of the measured values).

In order to evaluate the asynchronous search techniques, the message flow was counted; i.e. the quantity of ok and nogood messages exchanged by agents, the number of checked constraints, i.e. the local effort made by each agent, and the number of nonconcurrent constraints checks (defined in [10], noted with ncccs) necessary to obtain the solution.

In the AWCS family there are many variants that are based on building of efficient nogoods (nogood learning [9]) or on storing and using those nogoods in the process of selecting values (nogood processor). Two families of AWCS techniques are evaluated:

- basic variant proposed in [17] improved with the nogood learning technique (noted with AWCS-nl)
- basic variant proposed in [17] improved with the nogood learning technique and the nogood processor technique (noted with AWCS-nlng).

Four implementations are done corresponding to the obtained models:

- basic variant proposed in [17] improved with the nogood learning technique (noted with AWCS-nl)
- variant based on the nogood processor distributed to each agent: AWCS-nlng$_1$
- variant based on the nogood processor distributed only to the hub type agents: AWCS-nlng$_2$. This variant used the first method of message processing
- variant based on the nogood processor distributed only to the agents of the hub type: AWCS-nlng$_3$. This variant used the second solution of the message processing.

In the first stage, we evaluate four implementations in the conditions of running them on a cluster of computers using RedHat Linux. The methodology developed in [14] to run NetLogo models in a cluster computing environment allowed only running the model with synchronization based on the *ask-concurrent* command. We utilize the Java API of NetLogo as well as LoadLeveler. The solution without GUI allows to be run on a cluster of computers in the mode with synchronization,

as opposed to the GUI solution that can be run on a single computer and allows running in both ways: with synchronization or completely asynchronously.

The number of concurrent constraint checks (ncccs) allows to evaluate global effort without considering that the agents work concurrently (informally, the number of concurrent constraint checks approximates the longest sequence of constraint checks not performed concurrently). Analysing the results from Figure 4 a), one can remark that the method that distributes the nogood processors to the hub type nodes reduces the global effort made by the agents (AWCS-nlng$_2$ and AWCS-nlng$_3$). According to Definition 7, the number of hubs depends on the constant c. For the problems of the scale-free network type evaluated in this article, we considered as hub type nodes those having the degree over 60 % of the maximum degree. For the constraints graphs chosen previously, the number of hubs is approximatively as follows:

- nodes = 100, md = 16 and $c = 5$
- nodes = 500, md = 4 and $c = 3$
- nodes = 500, md = 16 and $c = 15$
- nodes = 1 000, md = 2 and $c = 4$

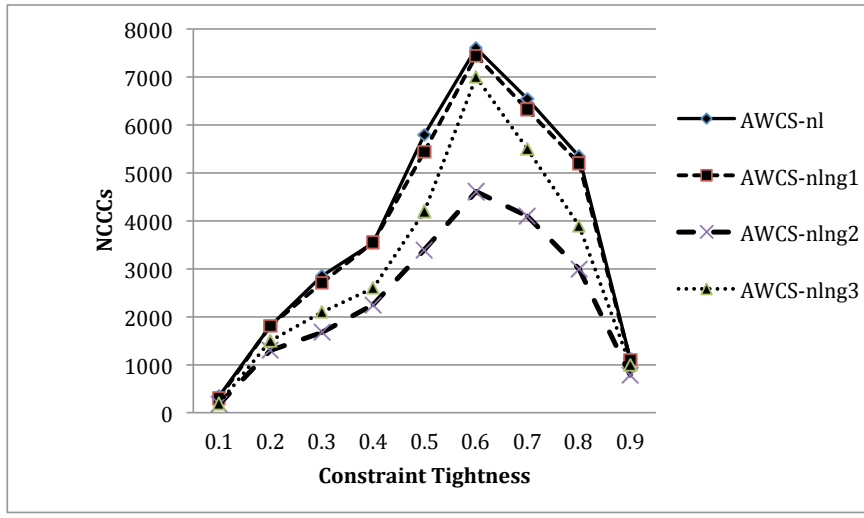
In case of instances with 1 000 agents, the four implementations had the same behaviour, the version AWCS-nlng$_3$ requiring the least effort for obtaining the solution.

In the case of the message flow (Figure 4 b)), one can notice almost equal efforts for obtaining the solution for all 4 versions. Though, for problems with a high difficulty (constraint tightness = 0.6) the AWCS-nlng$_3$ variant requires slightly less messages. Regarding the complete processing of the messages, experimental analysis shows that the first solution that completely processes the messages (AWCS-nlng$_2$) is better.

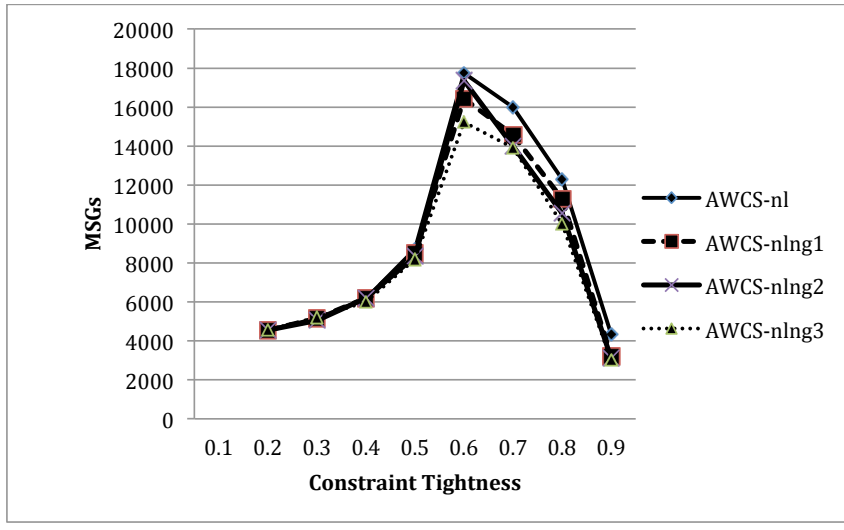
Another set of experiments was made for instances of scale-free networks with 500 nodes, but for which md is 16. Such instances are characterized also by a higher density of the constraints graph. Therefore, solving such instances presumes a much higher flow of messages. The results are presented in Figure 5. Analyzing the results from Figure 5, one can remark that the computing effort is pretty much the same, even if the versions AWCS-nlng$_3$ and AWCS-nlng$_2$ had lower costs but not significantly. A possible cause for that is the big number of hubs for the density of the constraints graph. A message flow almost identical can be observed for all the versions, the difference appearing only for the global effort (ncccs).

In the second stage we investigate the effect of the synchronization of the agents' execution process. The experiments were performed in the manner with GUI, on a single computer, with instances of 100 nodes. Seven implementations are done corresponding to the obtained models (this variants used the first method of message processing):

- variants based on synchronization with the aid of the "ask-concurrent" command (complete synchronization): AWCS-nl$_s$, AWCS-nlng$_s$, AWCS-nlnghub$_s$

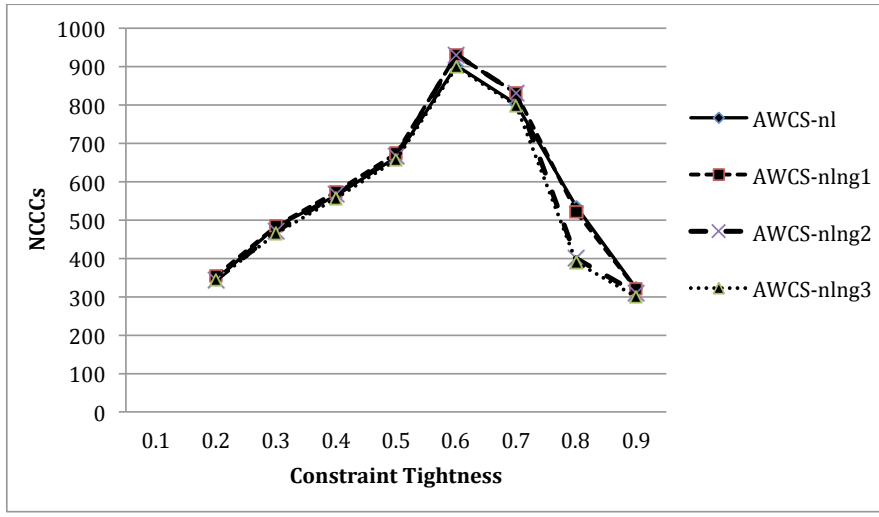


a)

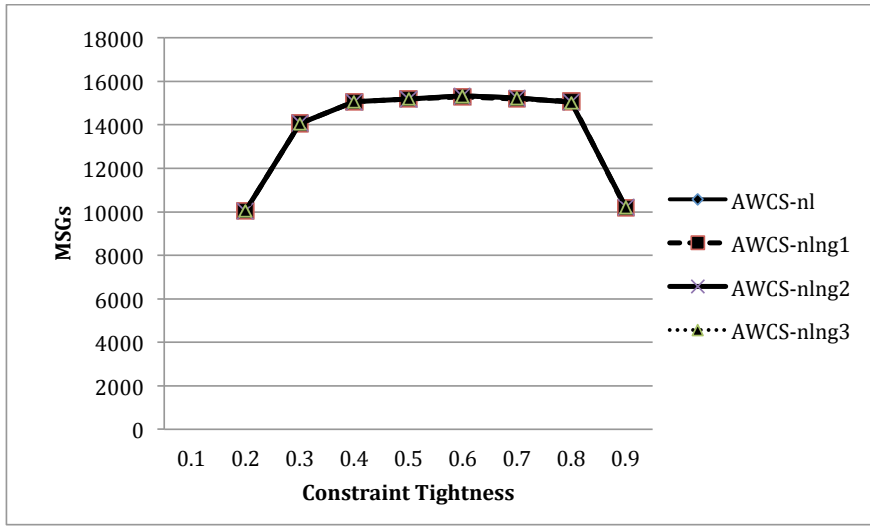


b)

Figure 4. Comparative study for the AWCS versions (scale-free networks), $n = 500$, $\mathrm{md} = 4$: a) the number of nonconcurrent constraint checks, b) total number of messages



a)



b)

Figure 5. Comparative study for the AWCS versions (scale-free networks), $n = 500$, $\text{md} = 16$: a) the number of nonconcurrent constraint checks, b) total number of messages

- variants based on the asynchronous model: AWCS-nl$_a$, AWCS-nlng$_a$, AWCS-nlnghub$_a$
- variant based on the second solution of synchronization (the partial synchronization of the agents): AWCS-nlnghub$_p$.

Results appear in Table 1, where we report the number of checked constraints (Constr.) the number of nonconcurrent constraint checks (Ncccs) and the total number of messages exchanged (Tmess), averaged over 1 000 executions.

n = 100 agents		md = 2		md = 16
		p2 = 0.4	p2 = 0.6	p2 = 0.2
AWCS-nl$_s$	TMess	741	1 140	11 040
	Constr.	2 100	3 156	5 480
	Ncccs	643	752	15 002
AWCS-nl$_a$	TMess	758	1 198	8 980
	Constr.	2 448	3 808	58 560
	Ncccs	986	1 439	15 722
AWCS-nlng$_s$	TMess	744	1 117	8 876
	Constr.	2 163	3 150	59 965
	Ncccs	673	732	15 332
AWCS-nlng$_a$	TMess	734	1 150	8 725
	Constr.	2 300	3 714	58 024
	Ncccs	914	1 413	15 496
AWCS-nlnghub$_s$	TMess	740	1 254	8 474
	Constr.	2 096	3 330	46 188
	Ncccs	646	765	15 006
AWCS-nlnghub$_a$	TMess	745	1 248	8 546
	Constr.	2 390	3 956	55 240
	Ncccs	946	1 489	15 126
AWCS-nlnghub$_p$	TMess	745	1 282	8 475
	Constr.	2 120	3 950	46 188
	Ncccs	724	776	15 117

Table 1. The results for AWCS versions (nodes = 100)

Analyzing the results from Table 1, one can notice that all seven implementations had approximately the same message exchange. Concerning the local or global effort of the agents, one can remark a difference between the asynchronous variants and those with a complete or partial synchronization. Surprisingly, the variants with synchronization required a lower effort.

In the case of problems with the high density (md = 16), the previous observations remain true.

The analysis of the first experiments shows that it is preferable to do a synchronization of the execution. The intermediate solution applicable in practice is that of the partial synchronization of the agents' execution.

6 CONCLUSIONS

In this paper we examine the effect of nogood processor for constraints networks of the scale-free type. We have shown that the manner of distributing the nogood values to agents depends on the number of connexions between agents in scale-free networks, thus obtaining a new hybrid search technique that uses the information from the stored nogoods.

We developed a novel way for the distribution of nogood values to agents, thus obtaining a new hybrid search technique that uses the information from the stored nogoods. The experiments show that it is more effective for several families of asynchronous techniques. Also, in this paper we examine the effect of synchronization of agents' execution and of a messages processing by packets in scale-free networks. The experimental analysis is performed in the situation of a model run on a cluster of computers and on a single computer.

We analyzed more versions obtained by distributing the nogood values to more nogood processors for each agent for constraints networks of a scale-free type.

The experimental analysis shows that the best way of distribution is that in which the nogood processors are distributed to nodes of the hub type (that have a very high degree, compared to other nodes). That variant requires the lowest costs.

Concerning the processing of messages by packets, the most performance versions were obtained in the case of treating all messages from the queue without inclusion of those that appeared later. Concerning the opportunity of synchronization of the agents' execution, the experiments show that in the case of the systems with synchronization the lowest costs were obtained. The compromise solution, called a partial synchronization, applied in practice required a lower computing effort than the solutions with entirely asynchronous running.

The combination of the analyzed elements (nogoods distribution, processing by packets and partial synchronization of the agents' execution) enabled us to identify a hybrid search technique with better performances for the case of problems with a free-scale network structure.

We believe that the combination proposed in this article could bring important benefits to the performances of the asynchronous techniques, leading to the reduction of an effort in finding the solution in the case of constraints networks of the scale-free type. Also, we want to extend the experiments for an even greater number of agents ($n > 1\,000$) and for other classes of problems with a scale free network structure, like for a random network.

REFERENCES

[1] Armstrong, A.—Durfee E.: Dynamic Prioritization of Complex Agents in Distributed Constraint Satisfaction Problems. Proceedings of the 15th IJCAI, Nagoya, Japan, 1997, pp. 620–625.

[2] Albert, R.—Barabási, A. L.: Statistical Mechanics of Complex Networks. Rev. Mod. Phys., Vol. 74, 2002, pp. 47–97.

[3] Barabási, A. L.—Albert, R.: Emergence of Scaling in Random Networks. Science, Vol. 286, 1999, pp. 509–512.

[4] Barabási, A. L.—Ravasz, E.—Vicsek, T.: Deterministic Scale-Free Networks. Physica A, Vol. 299, 2001, pp. 559–564.

[5] Barták, R.: Constraint Programming: In Pursuit of the Holy Grail. Proceedings of the Week of Doctoral Students (WDS), June 1999, pp. 555–564.

[6] Bessiere, C.—Brito, I.—Maestre, A.—Meseguer, P.: Asynchronous Backtracking without Adding Links: A New Member in the ABT Family. Artificial Intelligence, Vol. 161, 2005, pp. 7–24.

[7] Densmore, O.: An Exploration of Power-Law Networks. `http://backspaces.net/sun/PLaw/index.html`, 2009.

[8] Fernandez, C.—Bejar, R.—Krishnamachari, B.—Gomes, K.: Communication and Computation in Distributed CSP Algorithms. Proceedings of the Principles and Practice of Constraint Programming (CP-2002), Ithaca, NY, USA, July 2002, pp. 664–679.

[9] Hirayama, K.—Yokoo, M.: The Effect of Nogood Learning in Distributed Constraint Satisfaction. Proceedings of the 20th IEEE International Conference on Distributed Computing Systems (ICDCS-2000), 2000, pp. 169–177.

[10] Meisels, A.: Distributed Search by Constrained Agents: Algorithms, Performance, Communication. Springer Verlag, London, 2008, pp. 105–120.

[11] Muscalagiu, I.—Jiang, H.—Popa, H. E.: Implementation and Evaluation Model for the Asynchronous Techniques: From an Synchronously Distributed System to a Asynchronous Distributed System. Proceedings of the 8th SYNASC Conference, Timisoara, 2006, pp. 209–216.

[12] Muscalagiu, I.—Cretu, V.: Improving the Performances of Asynchronous Algorithms by Combining the Nogood Processors with the Nogood Learning Techniques. Journal "INFORMATICA", Lithuania, Vol. 17, 2006, No. 1.

[13] Muscalagiu, I.—Vidal, J.—Cretu, V.—Popa, H. E.—Panoiu, M.: The Efects of Agent Synchronization in Asynchronous Search Algorithms. Proceedings of the 1st KES Symposium on Agent and Multi-Agent Systems – Technologies and Applications, Springer-Verlag, LNAI, Vol. 4496, 2007, pp. 53–62.

[14] Muscalagiu, I.—Popa, H. E.—Vidal, J.: Clustered Computing with NetLogo for the Evaluation of Asynchronous Search Techniques. 12th International Conference on Intelligent Software Methodologies, Tools and Techniques, Budapest, Hungary, 2013.

[15] Muscalagiu, I.—Popa, H. E.—Negru, V.: The Impact of the "Nogood Processor" Technique in Scale-Free Networks. Proceedings of 7th International Symposium on Intelligent Distributed Computing (IDC 2013), September 4–6, 2013, Prague, Czech Republic, Springer, Studies in Computational Intelligence – Intelligent Distributed Computing VII, Vol. 511, 2013, pp. 163–173.

[16] Okimoto, T.—Iwasaki, A.—Yokoo, M.: Effect of DisCSP Variable-Ordering Heuristics in Scale-Free Networks. Multiagent and Grid Systems, Vol. 8, 2012, pp. 127–141.

[17] Yokoo, M.—Durfee, E. H.—Ishida, T.—Kuwabara, K.: The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. IEEE Transactions on Knowledge and Data Engineering, Vol. 10, 1998, No. 5, pp. 673–685.

[18] Yokoo, M.—Hirayama, K.: Algorithms for Distributed Constraint Satisfaction: A Review. Autonomous Agents and Multi-Agent System, Vol. 3, 2000, No. 2, pp. 198–212.

[19] Wilensky, U.: NetLogo Itself: NetLogo. Available on: `http://ccl.northwestern.edu/netlogo/`. Center for Connected Learning and Computer-Based Modeling, Evanston, 1999.

[20] MAS NetLogo Models-a. Available on: `http://discsp-netlogo.fih.upt.ro/`.

[21] InfraGRID Cluster. Available on: `http://hpc.uvt.ro/infrastructure/infragrid/`.

Ionel Muscalagiu received his M.Sc. degree in computer science from the West University of Timisoara in 1992 and his Ph.D. degree from the Politehnica University of Timisoara in 2008. He is working as Associate Professor in the Department of Electrical and Industrial Informatics, Politehnica University of Timisoara (Romania). Currently he is Head of the Department of Electrical and Industrial Informatics, Politehnica University of Timisoara. His areas of interest include distributed computing, constraint programming and multi-agent systems (distributed constraint satisfaction problem), and educational software. His research interests are focused on the distributed constraint; he is particularly interested in the issues of modeling, simulating and real execution of distributed constraints in NetLogo. He is author and co-author of many publications in the field.

Horia Emil Popa received his M.Sc. degree in computer science from the West University of Timisoara in 1992 and his Ph.D. degree from the same University in 2009. He is working as Lecturer in the Department of Informatics, Faculty of Mathematics and Computer Science of the West University of Timisoara (Romania). His areas of interest include recommendation systems, distributed computing, constraint programming and multi-agent system. His research interests are focused on the distributed constraint, he is particularly interested in the modeling, simulating and real execution of distributed constraints in NetLogo.

Viorel Negru is Full Professor at the Computer Science Department at the West University of Timisoara, Romania and Prorector for Research at the West University of Timisoara. He has published more than 130 papers in journals or conference volumes, most of them on artificial intelligence. He received his Ph.D. degree in computer science from the Babes-Bolyai University, Cluj-Napoca, Romania. His research interests include artificial intelligence, intelligent systems, intelligent front-end systems, multi-agent systems, data mining, and parallel and distributed computing.